MORE FILE ATTRIBUTES

Apart from permissions and ownership, a UNIX file has several other attributes, and in this chapter, we look at most of the remaining ones. A file also has properties related to its time stamps and links. It is important to know how these attributes are interpreted when applied to a directory or a device.

This chapter also introduces the concepts of file system. It also looks at the inode, the lookup table that contained almost all file attributes. Though a detailed treatment of the file systems is taken up later, knowledge of its basics is essential to our understanding of the significance of some of the file attributes. Basic file attributes has helped us to know about - ls –l to display file attributes (properties), listing of a specific directory, ownership and group ownership and different file permissions. ls –l provides attributes like – permissions, links, owner, group owner, size, date and the file name.

File Systems and inodes

The hard disk is split into distinct partitions, with a separate file system in each partition. Every file system has a directory structure headed by root.

n partitions = n file systems = n separate root directories

All attributes of a file except its name and contents are available in a table – inode (index node), accessed by the inode number. The inode contains the following attributes of a file:

- File type
- File permissions
- Number of links
- The UID of the owner
- The GID of the group owner
- File size in bytes
- Date and time of last modification
- Date and time of last access
- Date and time of last change of the inode
- An array of pointers that keep track of all disk blocks used by the file

Please note that, neither the name of the file nor the inode number is stored in the inode. To know inode number of a file:

```
ls -il tulec05
```

```
9059  -rw-r--r-- 1 kumar  metal  51813 Jan 31 11:15  tulec05
```

Where, 9059 is the inode number and no other file can have the same inode number in the same file system.

Hard Links

The link count is displayed in the second column of the listing. This count is normally 1, but the following files have two links,

```
-rwxr-xr--  2  kumar   metal   163   Jull   13 21:36   backup.sh
-rwxr-xr--  2  kumar   metal   163   Jul   13 21:36   restore.sh
```

All attributes seem to be identical, but the files could still be copies. It's the link count that seems to suggest that the files are linked to each other. But this can only be confirmed by using the –i option to ls.

```
ls -li backup.sh restore.sh
```

```
478274 -rwxr-xr-- 2 kumar metal163 jul 13 21:36 backup.sh
478274 -rwxr-xr-- 2 kumar metal163 jul 13 21:36 restore.sh
```

ln: Creating Hard Links

A file is linked with the ln command which takes two filenames as arguments (cp command). The command can create both a hard link and a soft link and has syntax similar to the one used by cp. The following command links emp.lst with employee:

```
ln emp.lst employee
```

The –i option to ls shows that they have the same inode number, meaning that they are actually one end the same file:

```
ls -li emp.lst employee
```

```
29518 -rwxr-xr-x 2 kumar metal 915 may 4 09:58 emp.lst
29518 -rwxr-xr-x 2 kumar metal 915 may 4 09:58 employee
```

The link count, which is normally one for unlinked files, is shown to be two. You can increase the number of links by adding the third file name emp.dat as:

```
ln employee emp.dat ; ls -l emp*
```

```
29518  -rwxr-xr-x  3  kumar  metal  915 may 4 09:58 emp.dat
29518  -rwxr-xr-x  3  kumar  metal  915 may 4 09:58 emp.lst
29518  -rwxr-xr-x  3  kumar  metal  915 may 4 09:58 employee
```

You can link multiple files, but then the destination filename must be a directory. A file is considered to be completely removed from the file system when its link count drops to zero. ln returns an error when the destination file exists. Use the –f option to force the removal of the existing link before creation of the new one

Where to use Hard Links

ln data/ foo.txt input_files

It creates link in directory *input_files*. With this link available, your existing programs will continue to find foo.txt in the *input_files* directory. It is more convenient to do this that modifies all programs to point to the new path. Links provide some protection against accidental deletion, especially when they exist in different directories. Because of links, we don't need to maintain two programs as two separate disk files if there is very little difference between them. A file's name is available to a C program and to a shell script. A single file with two links can have its program logic make it behave in two different ways depending on the name by which it is called.

We can't have two linked filenames in two file systems and we can't link a directory even within the same file system. This can be solved by using symbolic links (soft links).

Symbolic Links

Unlike the hard linked, a symbolic link doesn't have the file's contents, but simply provides the pathname of the file that actually has the contents.

ln -s note note.sym

ls -li note note.sym

9948 -rw-r--r-- 1 kumar group 80 feb 16 14:52 note
9952 lrwxrwxrwx 1 kumar group 4 feb16 15:07note.sym ->note

Where, l indicate symbolic link file category. -> indicates note.sym contains the pathname for the filename note. Size of symbolic link is only 4 bytes; it is the length of the pathname of note.

It's important that this time we indeed have two files, and they are not identical. Removing note.sym won't affect us much because we can easily recreate the link. But if we remove note, we would lose the file containing the data. In that case, note.sym would point to a nonexistent file and become a dangling symbolic link.

Symbolic links can also be used with relative pathnames. Unlike hard links, they can also span multiple file systems and also link directories. If you have to link all filenames in a directory to another directory, it makes sense to simply link the directories. Like other files, a symbolic link has a separate directory entry with its own inode number. This means that rm can remove a symbolic link even if its points to a directory.

A symbolic link has an inode number separate from the file that it points to. In most cases, the pathname is stored in the symbolic link and occupies space on disk.

However, Linux uses a fast symbolic link which stores the pathname in the inode itself provided it doesn't exceed 60 characters.

The Directory

A directory has its own permissions, owners and links. The significance of the file attributes change a great deal when applied to a directory. For example, the size of a directory is in no way related to the size of files that exists in the directory, but rather to the number of files housed by it. The higher the number of files, the larger the directory size. Permission acquires a different meaning when the term is applied to a directory.

```
ls -l -d progs
```

```
drwxr-xr-x  2  kumar  metal 320 may 9 09:57 progs
```

The default permissions are different from those of ordinary files. The user has all permissions, and group and others have read and execute permissions only. The permissions of a directory also impact the security of its files. To understand how that can happen, we must know what permissions for a directory really mean.

Read permission

Read permission for a directory means that the list of filenames stored in that directory is accessible. Since ls reads the directory to display filenames, if a directory's read permission is removed, ls wont work. Consider removing the read permission first from the directory *progs*,

```
ls -ld progs
```

```
drwxr-xr-x  2  kumar  metal 128 jun 18 22:41 progs
```

```
chmod -r progs ; ls progs
```

```
progs: permission denied
```

Write permission

We can't write to a directory file. Only the kernel can do that. If that were possible, any user could destroy the integrity of the file system. Write permission for a directory implies that you are permitted to create or remove files in it. To try that out, restore the read permission and remove the write permission from the directory before you try to copy a file to it.

```
chmod 555 progs ; ls –ld progs
```

```
dr-xr-xr-x 2 kumar metal 128 jun 18 22:41 progs
```

cp emp.lst progs

cp: cannot create progs/emp.lst: permission denied

- The write permission for a directory determines whether we can create or remove files in it because these actions modify the directory
- Whether we can modify a file depends on whether the file itself has write permission. Changing a file doesn't modify its directory entry

Execute permission

If a single directory in the pathname doesn't have execute permission, then it can't be searched for the name of the next directory. That's why the execute privilege of a directory is often referred to as the search permission. A directory has to be searched for the next directory, so the cd command won't work if the search permission for the directory is turned off.

chmod 666 progs ; ls –ld progs

drw-rw-rw- 2 kumar metal 128 jun 18 22:41 progs

cd progs

permission denied to search and execute it

umask: DEFAULT FILE AND DIRECTORY PERMISSIONS

When we create files and directories, the permissions assigned to them depend on the system's default setting. The UNIX system has the following default permissions for all files and directories.

rw-rw-rw- (octal 666) for regular files

rwxrwxrwx (octal 777) for directories

The default is transformed by subtracting the user mask from it to remove one or more permissions. We can evaluate the current value of the mask by using *umask* without arguments,

$ umask
022

This becomes 644 (666-022) for ordinary files and 755 (777-022) for directories *umask* 000. This indicates, we are not subtracting anything and the default permissions will remain unchanged. Note that, changing system wide default permission settings is possible using *chmod* but not by *umask*

MODIFICATION AND ACCESS TIMES

A UNIX file has three time stamps associated with it. Among them, two are:

- Time of last file modification ls -l
- Time of last access ls –lu

The access time is displayed when ls -l is combined with the -u option. Knowledge of file‘s modification and access times is extremely important for the system administrator. Many of the tools used by them look at these time stamps to decide whether a particular file will participate in a backup or not.

TOUCH COMMAND – changing the time stamps

To set the modification and access times to predefined values, we have,

touch options expression filename(s)

touch emp.lst (without options and expression)

Then, both times are set to the current time and creates the file, if it doesn’t exist.

touch command (without options but with expression) can be used. The expression consists of MMDDhhmm (month, day, hour and minute).

touch 03161430 emp.lst ; ls -l emp.lst

-rw-r--r-- 1 kumar metal 870 mar 16 14:30 emp.lst

ls -lu emp.lst

-rw-r--r-- 1 kumar metal 870 mar 16 14:30 emp.lst

It is possible to change the two times individually. The –m and –a options change the modification and access times, respectively:

touch command (with options and expression)

-m for changing modification time
-a for changing access time

touch -m 02281030 emp.lst ; ls -l emp.lst

-rw-r--r-- 1 kumar metal 870 feb 28 10:30 emp.lst

touch -a 01261650 emp.lst ; ls -lu emp.lst

-rw-r--r-- 1 kumar metal 870 jan 26 16:50 emp.lst

find : locating files

It recursively examines a directory tree to look for files matching some criteria, and then takes some action on the selected files. It has a difficult command line, and if you have ever wondered why UNIX is hated by many, then you should look up the cryptic *find* documentation. How ever, *find* is easily tamed if you break up its arguments into three components:

find path_list selecton_criteria action

where,

- Recursively examines all files specified in path_list
- It then matches each file for one or more selection-criteria
- It takes some action on those selected files

The path_list comprises one or more subdirectories separated by white space. There can also be a host of selection_criteria that you use to match a file, and multiple actions to dispose of the file. This makes the command difficult to use initially, but it is a program that every user must master since it lets him make file selection under practically any condition.

- Source: Sumitabha Das, "UNIX – Concepts and Applications", 4th edition, Tata McGraw Hill, 2006